DoWitcher: Effective Worm Detection and Containment in the Internet Core

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE INFOCOM 2007 proceedings.

DoWitcher: Effective Worm Detection and Containment in the Internet Core

S. Ranjan[1], S. Shah[1], A. Nucci[1], M. Munafò[2], R. Cruz[3], S. Muthukrishnan[4]
[1] – Narus Inc, Mountain View, CA, USA
[2] – Politecnico di Torino, Turin, Italy
[3] – Univ California San Diego, CA, USA
[4] – Rutgers Univ, NJ, USA

***Abstract*—Enterprise networks are increasingly offloading the responsibility for worm detection and containment to the carrier networks. However, current approaches to the zero-day worm detection problem such as those based on content similarity of packet payloads are not scalable to the carrier link speeds (OC-48 and up-wards). In this paper, we introduce a new system, namely DoWitcher, which in contrast to previous approaches is scalable as well as able to detect the stealthiest worms that employ low-propagation rates or polymorphisms to evade detection. DoWitcher uses an incremental approach toward worm detection: First, it examines the layer-4 traffic features to discern the presence of a worm anomaly; Next, it determines a flow-filter mask that can be applied to isolate the suspect worm flows and; Finally, it enables full-packet capture of only those flows that match the mask, which are then processed by a Longest Common Subsequence algorithm to extract the worm content signature. Via a proof-of-concept implementation on a commercially available network analyzer processing raw packets from an OC-48 link, we demonstrate the capability of DoWitcher to detect low-rate worms and extract signatures for even the polymorphic worms.**

I. Introduction

Recent years have seen a sharp increase in Internet worms causing damage to millions of systems worldwide. Worms are automated programs that exploit vulnerabilities in computers connected to the network in order to gain control over them. Once they have successfully infected a system, they continue to search for new victims and can spread through the network on their own. Each new epidemic has demonstrated a greater speed and virulence over its predecessor. While the Code Red worm took over fourteen hours to infect its vulnerable population in 2001, the Slammer worm, released some 18 months later, did the same in under 10 minutes [1]. The Code Red worm is thought to have infected roughly 360,000 hosts, while by some estimates, the Nimda worm compromised over two million.

However, the next generation of worms can be expected to spread much slower than its predecessors as well as employ polymorphic techniques by either obfuscating or encrypting the worm payload. With this increasing sophistication in worms, enterprise networks can no longer rely on signature-based solutions such as Intrusion Detection or Prevention Systems (IDS/IDS) to promptly detect a new worm incidence, popularly referred to as "zero-day worms". Moreover, the traffic volume of a zero-day worm purposely designed to be low-rate would be "below-the-radar" of volume anomaly detection techniques.

These sophisticated zero-day worms can not be promptly detected via techniques deployed at an enterprise network which has a limited visibility in to only the traffic entering or leaving its network perimeter. Infact, carrier networks are in a unique position to promptly detect as well as perform forensics on such worm attacks. This is due to the fact that carrier networks are afforded a natural correlation since they see traffic from their customers via the edge-links as well as from other carriers via the peering links, leading to a *total network view*. As a consequence, we envision that in the near future, enterprises would shift their security burden to carriers where the deployment of a comprehensive security shield would be much more efficient and proactive.

However, most existing worm detection algorithms have not been designed with the unique performance requirements of a carrier network in mind. While they have shown good performance in detecting worms at the speeds of enterprise networks e.g., 100 Mbps to 1 Gbps, they are not scalable to the high data rate links which characterize the carrier networks such as OC-48, OC-192 and up-wards. The past methods that have been proposed to identify new worms can be divided into two major categories.

The first class is based on content fingerprinting using the layer-7 traffic information [2]–[4]. The primary intuition underlying this class is that an ongoing worm propagation should manifest itself in the presence of higher than expected *byte-level similarity* among network packets: the similarity arises because of the unchanging portions of the worm packet payload, something expected to be present even in polymorphic or obfuscated worms (albeit spread out over the length of the packet or across several packets belonging to the flow).

In contrast to, the second class [5]–[7] consists of techniques which identify network anomalies by examining the layer-4 traffic distribution across a few features. The primary intuition underlying these approaches is that a worm manifestation breaks the statistical characteristics of Internet traffic; worm traffic is more uniform or structured than normal traffic in some respects and more random in others. Approaches such as [7] and [6] propose techniques based on Principle Component Analysis (PCA) and Residual State Analysis (RSA) respectively, to establish complex relationships between the traffic features using which flows are classified as either legitimate or malicious. However, we contend that these ap-

0743-166X/07/$25.00 ©2007 IEEE

proaches, while robust, are primarily offline and hence not effective for worm containment at the high data rate links atypical of the Internet core.

In this paper, we bridge the gap between the two classes by proposing a novel worm anomaly detection system, called *DoWitcher*, that brings together the characteristics of being deployable for real-time high data rate links and the efficiency and reliability of content fingerprinting techniques. The scalability achieved by DoWitcher is based on its two-tiered functional architecture, where initially anomaly detection is performed using only the layer-4 information from the packets and on determining the presence of an anomaly, the layer-7 payloads of only the anomalous flows are compared to extract the worm signature. Dowitcher's statistical anomaly detection is based on the observation that during a worm outbreak at least two of the traffic features exhibit diverging behaviors. Further, signature extraction is achieved by comparing the payloads of only the flows that match a layer-4 filter mask via a Longest Common Subsequence (LCS) algorithm.

II. DoWitcher System Architecture

DoWitcher is a highly scalable system able to collect raw packets from network links and router interfaces, locally extract key traffic features and then process all meta-data in one single central location, critical for generating a real-time network-wide view of traffic activity. The system is composed of two modules: *Local* and *Global* analyzers.

The DoWitcher Local Analyzer (DLA), shown in Figure 1(a), represents the interface with network elements that collects raw packets through a *Network Tap*, e.g., passive wire tap to collect packets off the wire or port mirroring to collect packets directly from router interfaces. The packets are then processed by the *Flow Reconstruction* module that buffers packets, reorders the out-of-sequence packets and state-fully reconstructs the flows in real-time. A flow is defined by all the packets identified by the layer-4 five-tuple, in between and including the SYN and FIN packets (in case of TCP) or, until there is no packet arrival within a specified timeout. The flows are then processed by the *Classifier* module that extracts key features from each flow.

At the end of a time window, DLAs deployed across the network forward this information to the DoWitcher Global Analyzer (DGA), shown in Figure 1(b). The DGA is in charge of several functions: (i) extracting the histograms of the key flow features, and summarizing their properties by computing their entropies (Histogram Extraction Module); (ii) grouping all entropies together in one single efficient metric, (PMER Computation); (iii) profiling the metric over time and generating alerts when an instantaneous deviation from the historical trend of the current metric is observed (Baseline and Alerting Module); (iv) composing a policy rule that captures the worm activity from a traffic flow perspective (Flow Filter Mask Generation). The policy is then forwarded down to all DLAs. A typical policy would request capturing raw packets from a few infected end hosts, using a specific destination or source port and a fixed or range of flow sizes. As soon as the policy is received, each local analyzer starts *full packet capture* of packets belonging to flows matching the policy rule. After the flows are state-fully reconstructed, the *Flow Payload Extraction* module strips out the packet headers and stores only their payloads, generating what is defined as the *flow payload content*. Finally, this information is forwarded to the the DGA that extracts the worm signature by comparing the payload contents.

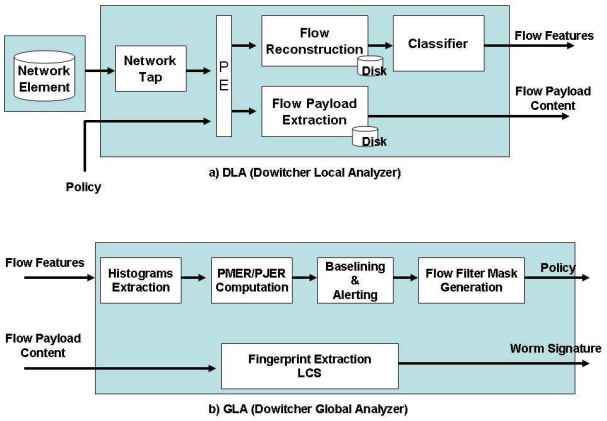


Fig. 1. Logical Architecture of DoWitcher: local and global analyzers.

III. Algorithm

In this section, we describe in great details the major functionalities of our system, by emphasizing the Classifier (DLA module) as well as all the blocks that comprise the DGA described in Figure 1.

A. Classifier and Histogram Extraction Module

Due to its inherent goal to spread widely and quickly, an effective worm might not affect the traffic volume yet while severely disrupting the *normal structure* of Internet traffic due its intrinsic characteristics to be more uniform or structured than normal traffic in some respects and more random in others. Usually, a small number of hosts try to find other hosts to infect by attempting to connect to them in a purely random fashion, scanning for a specific vulnerability. Moreover, Internet worm payloads tend to be small in size in order to spread as fast as possible. A large worm size would prolong infection time and consume bandwidth that could be used for infecting other targets.

Thus, DoWitcher extracts the following *flow* traffic features whose histograms we will show to be affected during the spread of the worm: (i) *source ip-address*, (ii) *source port*, (iii) *destination ip-address*, (iv) *destination port* and (v) *flow-size*. For example, a shift in the *source ip-address histogram* will be expected whenever the number of flows generated by the infected hosts grows to be a significant part of the total observed flows; the source ip-addresses of the scanning hosts will be seen in many flows and the distribution of the source IP address will become more skewed around the few infected hosts. A similar thing happens on the *source port and destination port histograms*. If any attacker scans

for a specific vulnerability, these scans often have to go to a specific target destination port. The source ports of these connections are usually selected in some weakly random fashion from a range of possible source ports. Example of these behaviors are visible in worms seen in the past, like the *Sapphire* (destination-port 1434), *CodeRed* (destination-port 80), Welchia (destination-port 135) and many others. A few worms, like the *Witty*, behave unexpectedly, by using a fixed source port and variable destination port. Similar consideration holds for the *flow-size histogram*. If a specific flow size becomes a significant component of the overall traffic, the distribution of the flow size will become more skewed around the flow size used by the worm. On the other hand, the *destination ip-address histogram* can be expected to become more flat due to the inherent random scanning activity of the destination hosts.

A typical carrier link can consist of hundreds of thousands of distinct ip-addresses at any given time. Hence, it may be prohibitive to construct histograms over distinct ip-addresses. DoWitcher's histogram extraction is designed with the goal of providing memory savings by aggregating flows over clusters of ip-addresses. Clusters might be defined either *statically*, as predefined blocks of IP addresses i.e., subnets (/16, /24) or, *dynamically*, as groups of hosts behaving similarly over time in terms of traffic.

B. PMER Computation Module

In order to track changes in the shape of the featured histogram, we introduce the concept of *Entropy* that measures how random a data-set is; the more random it is, the more entropy it contains. In the following, we mathematically introduce the concept of entropy. Let's assume we are monitoring a generic key feature X over time for a specific cluster A and let $M^X(x)$ be its frequency distribution i.e., number of times we see an element $x \in X$. From the frequency distribution $M_i^X(x) = \{x_i\}$ in time window i, we can derive the empirical probability distributions $P_i^X(x)$ as: $P_i^X(x) = \{p_i^X | p_i^X = \frac{x_i}{m^X}\}$, where $m^X = \sum x_i$ is the overall number of flows that contributed to the distribution during time window i.

From this probability distribution P_i^X we calculate the information entropy H_i^X as: $H_i^X = -\sum_{p \in P_i^X} p \log_2 p$, where by convention $0 \log_2 0 = 0$. As known, entropy will be an indication of the uniformity of the distributions: low entropy indicates high probability in few elements (concentrated usage of the same port, high traffic from the same source), while high entropy indicates a more uniform usage (random scan of destination IP, variable source port). Since we are using $\log_2(\cdot)$ in our definition, each H_i^X will assume values in a range between 0 and $\lceil \log_2(N^X) \rceil$, N^X being the maximum number of distinct values X can assume in the time window. In order to have a metric H_i^X independent of the number of distinct values, we normalize the Entropy by the size of its support, i.e., $\lceil \log_2(N^X) \rceil$. This normalized Entropy is also known in literature as *Relative Uncertainty (RU)* and in the paper we refer to this metric as H^X.

1) Single Global Metric: DoWitcher is based on the hypothesis that during a worm outbreak, the Relative Uncertainties of at least two of the five features will diverge [5]. In order to capture this dynamics, we propose in this paper a novel metric, namely *Pair-Wise Marginal Entropy Ratio*, that exhibits a stable behavior under normal conditions, while a sharp increase even during a low volume worm or scan activity.

First, let F denote the set of features (in this case, $|F| = 5$), and let (X, Y) denote a pair of features in F such that $X, Y \in F, X \neq Y$. Further, let $R_i^{XY} = \frac{H_i^X}{H_i^Y}$ represent the instantaneous ratio between the two marginal RUs H^X and H^Y and $\text{avg}(R_i^{XY}) = 1/N_S \sum_{k=i-N_S}^{i-1} \frac{H_k^X}{H_k^Y}$ represent the average value of R^{XY} over the last N_S time-windows.

Definition 1: Pair-Wise Marginal Entropy Ratio (PMER) is defined as the maximum over all feature-pairs (X, Y) of the ratio between the marginal RUs (H^X and H^Y) and its average computed using the last N_S time-windows. Then, PMER can be computed as: $R_i = \max_{(X,Y)} |\frac{R_i^{XY}}{\text{avg}(R_i^{XY})} - 1|$.

At each point in time i, the metric captures the shift in the slope between the instantaneous value of R^{XY} and its averaged value $\text{avg}(R^{XY})$ for all feature-pairs. Thus, *PMER* monitors the maximum divergence from normal behavior across all possible feature-pairs. Note that the *PMER* metric is space- and memory-efficient since it uses only marginal histograms for its computation. However, since *PMER* does not have information about joint-distributions, to accurately identify the worm flows, we need an additional mechanism, a flow filter-mask.

C. Baselining and Alerting Module

Our anomaly detection consists broadly of two phases: offline baselining and; on-line detection. First, we perform an offline characterization of traffic from system logs, assuming that the traffic consists of legitimate flows solely i.e., it is uninfluenced by any worms propagating through the network. Next, in an on-line phase, we compare the ongoing traffic's statistics with the legitimate profiles obtained previously. We thus define the anomaly detection problem in generic terms, and propose a new method, namely *forecasting* which uses a weighted average of traffic statistic in the past few observations, using which it forecasts the statistic for the next time interval and flags the traffic as suspicious if it deviates significantly from the historical trend. In the following we mathematically introduce the forecasting profiling algorithm for the PMER metric:

1) Define an averaging set $\mathcal{R}_i$ with $|\mathcal{R}_i| = W$ containing the last $W > N_S$ R_j samples considered being *in profile*. During baselining all the R_j are considered *in profile* and included in $\mathcal{R}_i$. $\mathcal{R}_i$ is valid only for $i \geq W$, so we need to collect data for at least W time windows before being able to run the algorithm.
2) Maintain a running average of R_i over $\mathcal{R}_i$ as: $\overline{R}_i = \frac{1}{W} \sum_{R_n \in \mathcal{R}_i} R_n$. During baselining, since all the last W R_i samples are in $\mathcal{R}_i$, this becomes: $\overline{R}_i = \frac{1}{W} \sum_{n=i-W+1}^{i} R_n$.
3) Define control coefficients $\hat{\alpha}_i^{\max}$ and let $\hat{\alpha}_i^{\max} = \hat{\alpha}_i^{\max} = 1$ for $0 < i < W$.

4) For $W < i \leq T_w$, T_w being the length of the baselining period: if $R_i > \hat{\alpha}^{\max}_{i-1}\overline{R}_{i-1}$ then $\hat{\alpha}^{\max}_i = \frac{R_i}{\overline{R}_{i-1}}$ else $\hat{\alpha}^{\max}_i = \hat{\alpha}^{\max}_{i-1}$.

At the end of the baselining period, $\hat{\alpha}^{\max}_{T_w}$ will contain the largest measured excursion between one sample and the average $\overline{R}$ value in the near past. We freeze this value, defining $\alpha^{\max} = \hat{\alpha}^{\max}_{T_w}$.

After the baselining period, we will report an anomaly whenever $R_i > \alpha^{\max}\overline{R}_{i-1}$. If no anomaly is revealed, the sample R_i is added to $\mathcal{R}_i$, dropping the oldest value in the set, otherwise the sample is discarded, $\mathcal{R}_i$ does not change and the average is not updated. The set for the calculation of the running average $\overline{R}$ is therefore detached from the samples coming from the measurements and it will contain only *good* measures.

D. Flow-Filter Mask Generation Module

As soon as an alert is raised, DoWitcher analyzes the cause of the deviation and identifies which features were involved in the anomaly i.e., sudden change in their marginal RUs. At this point, the algorithm focuses on the features for which it notices a decrease in their marginal RU values between the previous and the current time-window i.e., features whose histograms become suddenly concentrated around specific elements. The elements contributing the most to the decrease of the RU are identified using the concept of *relative entropy*, defined in the following. Let's assume that at a specific point in time i we detect an anomaly from one of the monitored cluster, according to the forecasting approach. We remind the reader that this condition is an alert if and only if two of the five marginal RUs deviate in the opposite direction. Let's assume X be one of the key features that exhibits a decrease in its RU value from time-window $i-1$ to i, i.e. $H^X_i < H^X_{i-1}$. At this point we need to identify the set $L_{X_i} = \{x_i \in M^X_i(x)\}$ that are contributing the most to the decrease. The cardinality of the set L_{X_i} is defined as an input of the algorithm and it is represented by $|L_{X_i}|$. Given the empirical probability distributions $P^X_i(x)$ and $P^X_{i-1}(x)$, we compute the relative entropy RE^x_i for each of the elements $\{x_i \in M^X_i(x)\}$ as following:

$$\mathrm{RE}^x_i = (p^X_i / \lceil \log_2(N^X_i) \rceil) log_2 \frac{p^X_i / \lceil \log_2(N^X_i) \rceil}{p^X_{i-1} / \lceil \log_2(N^X_{i-1}) \rceil}. \quad (1)$$

Then we sort the $\{x_i\}$ according to their relative entropy value RE^x_i and we select store in the set L_{X_i} the elements contributing the most to RE^x_i. The cardinality of L_{X_i} depends on the amount of information required for further analysis. Note that, for features that do not exhibit a drop at time i, we define $L_{X_i} = \emptyset$.

By applying the above procedure to all key features at time i, we generate a set of flow-filter masks $M_i = \{m_i\}$, where a filter mask m_i is obtained by an intersection of the elements of each feature's set of top elements, L or, $m_i = \bigcap_{X \in F} L_{X_i}$. Thus, for instance, if a source prefix 10.10.10.0/24 is detected to originate a Slammer worm, then the filter mask could look as follows: "source ip-address=10.10.10.0/24, destination ip-address=*, source port=*, destination port=1434, flow size=404" which when matched would return all Slammer worm flows originating from the infected prefix.

E. Fingerprinting

After extracting the payloads of the flows that match the flow-filter mask, the payloads of two flows are compared to obtain the worm fingerprint.

The fingerprint extraction algorithm compares the payloads to extract the complex signatures associated with the obfuscating strategies employed by *polymorphic worms*. These worms have the characteristics to change their signatures during their propagation into the network, by introducing wildcards or completely random characters, or by encrypting the payload while hiding the decryption key somewhere around the payload. However, even the polymorphic worms have an invariant across different flows. We pose this problem as determining the longest common subsequence between two strings. A string s is said to be a subsequence of string S, if s can be obtained by deleting 0 or more characters from string S. Thus, a string s is a longest common subsequence of strings S and T, if s is a common subsequence of S and T and there is no other common subsequence of S and T of greater length. For example, if two packet payloads contain the following strings h**ou**seboa**t** and c**o**mp**ut**er, the longest common subsequence that LCS reports is `out`. In contrast, an algorithm that looks for the common substring, will not report any commonality between the two packet payloads.

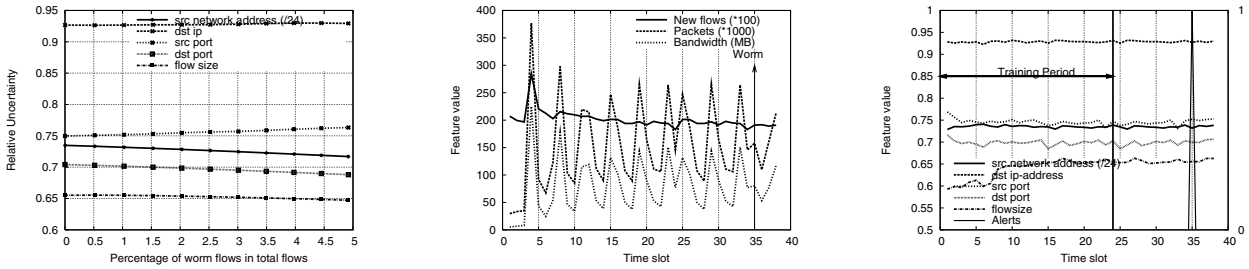
We use a simple dynamic programming approach toward solving the LCS problem, that has polynomial complexity $O(nm)$, where n and m represent the length of the two strings A and B in tokens. A token is a contiguous set of characters i.e., a substring which is by default set to one character. For each of the m starting points of A, the algorithm checks for the longest common subsequence of tokens starting at each of the n starting points of B.

IV. Testbed Implementation and Results

In this section, we present details of our testbed implementation and results that justify the efficiency and efficacy of DoWitcher in detecting low-rate as well as polymorphic worms. In our experiments, we use network traces obtained from one of the largest Tier-1 carrier networks in South America.

To validate the efficacy of DoWitcher, we use a testbed consisting of two machines, connected via an OC-48 link (2.4 Gbps). Each machine has two Intel(R) Xeon(TM) CPU 3.40GHz processors and 4 Gbytes memory available to a process running on the Linux kernel 2.4.21. We replay traces by using *tcpreplay* from one machine while the other one is configured to sniff packets off-the-wire and runs our packet analyzer, flow reconstruction, anomaly detection and full packet capture modules.

Next, we present results which highlight the efficacy of DoWitcher in detecting even the low propagation rate worms, which are un-detectable via volume metrics of total bandwidth, total packets or number of new flows. In this experiment, we use a 20 minute network trace from a large South American carrier network and use a time window of 30 seconds for



(a) Change in RU (b) Volume metrics (c) Behavior metric (PMER)

Fig. 2. Subfigure (a) shows the changes in RU on introduction of worm flows. Subfigure (b) shows that the worm doesn't introduce any change in total traffic volume while (c) shows its detection via PMER metric.

calculation of the histograms and the PMER metric. The baselining period T_w of the PMER metric lasts until window number 24, while the PMER is calculated by using the following values for the sliding windows: $W = 5$ and $N_S = 5$.

We inject a varying proportion of synthetic CodeRed worm traffic in to the trace around time window 35 according to the following worm propagation model: One ip-address, belonging to one of the most-widely used subnet is assumed to be infected. This infected machine then generates z worm flows (TCP) per window destined to port 80. The infected host performs a network scan and initiates one TCP flow per destination ip-address, where the destination ip-addresses are chosen as follows: 50% of the addresses contacted did not occur in the last time window while the rest were seen in the last window. The source port is chosen randomly and each worm flow consists of a total of 15 packets and a total flow-size of 4156 bytes, including the TCP handshake and tear-down packets. In our experiments, we increase the number of worm flows generated in time window 35 as $z = [0\text{-}1000]$ until the worm is detected via our PMER metric based alerting. Observe from Figure 2(a), that the RU for source network address, flow-size and destination-port decrease on increasing the proportion of worm flows, since the infected host, flow-size of 4156 and destination port of 80 begin to dominate the histograms. In contrast, the RU for destination ip-address and source-port do not exhibit much changes.

The worm becomes detectable via the PMER metric when there are 200 worm flows per time window as shown in Figure 2(c). This is indeed a low propagation rate worm, since it comprises of 1.0% flows, 2.0% packets and 1.1% bytes of total traffic in that time window, and is indistinguishable from clean traffic completely due to its low volume as shown in Figure 2(b).

In another experiment, we changed the payload of one of the worm flows by spreading the content across multiple packets and inserting arbitrary *no operator* or 'nop' characters in the worm payload executable. In each of the cases, our LCS algorithm was able to extract the CodeRed signature as `GET./default.ida?` and most importantly without imposing any significant CPU or memory performance overheads.

V. Conclusions

In this paper, we presented DoWitcher, a novel system for worm detection and containment at the carrier network link speeds (OC-48, OC-192 and up-wards). DoWitcher provides the capability of detecting zero-day worm threats by analyzing the key features of traffic flows, isolating the suspect worm flows via the generation of a flow filter-mask and extraction of the worm content signature via a LCS algorithm applied over the flow payload content of isolated flows. Thus, in contrast with previous approaches, DoWitcher derives its scalability by this two-tiered approach where only the layer-4 traffic features are used for worm detection and the fingerprint extraction is done on only a few suspect flows. Through testbed experiments, we established the efficacy and efficiency of DoWitcher in detecting the more sophisticated stealth worms: low propagation rate worms which utilize as low as 1.1% of the total network bandwidth and; polymorphic worms which spread their signature across multiple packets.

References

[1] D. Moore, V. Paxon, S. Savage, C. Shannon, S. Staniford, and N. Weaver "The Spread of the Sapphire/Slammer Worm", *IEEE Security and Privacy, 1(4)* July 2003

[2] S. Singh, C. Estan, G. Varghese, and S. Savage "Automated Worm Fingerprinting," *Proceedings of the 6th ACM/USENIX Symposium on Operating System Design and Implementation (OSDI),* December 2004

[3] H.-A. Kim and B. Karp, "Autograph: toward automated, distributed worm signature detection," *Proceedings of the 13th USENIX Symposium,* August 2004

[4] J. Newsome, B. Karp, and D. Song, "Polygraph: Automatically Generating Signatures for Polymorphic Worms,", *Proceedings of IEEE Security and Privacy Symposium,* Oakland, CA USA, May 2005

[5] A. Wagner and B. Plattner "Entropy Based Worm and Anomaly Detection in Fast IP Networks", *IEEE 14-th International Workshop on Enabling Technologies: Infrastructures for Collaborative Enterprises (WET ICE), STCA security workshop* June 2005

[6] A. Lakhina, M. Crovella, and C. Diot "Mining Anomalies Using Traffic Feature Distributions", *ACM Sigcomm* August 2005

[7] K. Xu, Z. Zhang and S. Bhattacharyya "Profiling Internet Backbone Traffic: Behavior Models and Applications", *ACM Sigcomm* August 2005